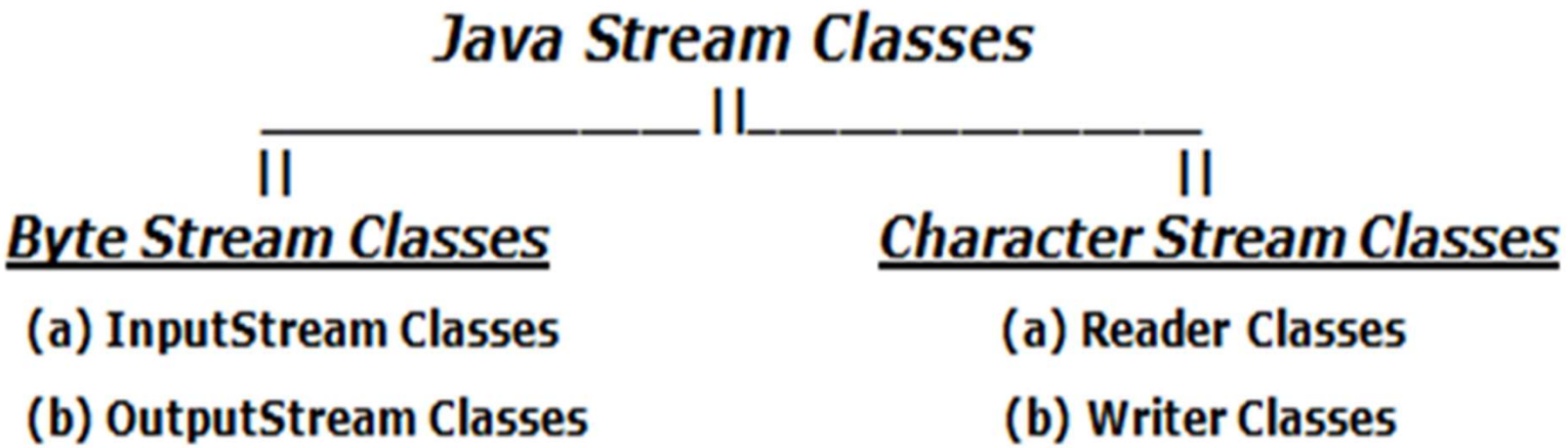
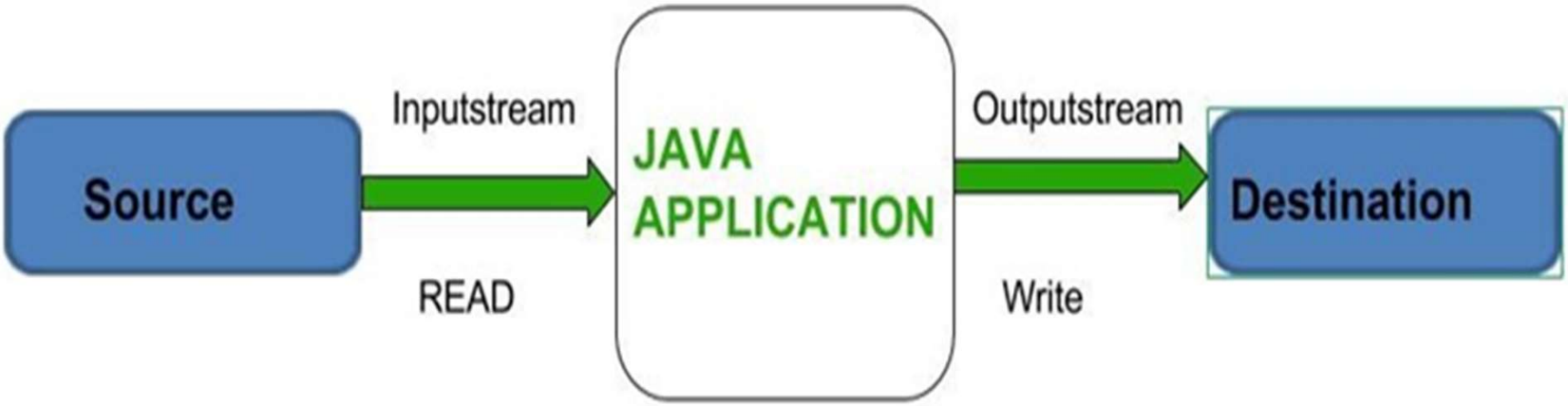
java.io package

Stream classes in Java

- In Java, a stream is a path along which the data flows.
- Every stream has a source and a destination.
- The InputStream is used to read data from a source and the OutputStream is used for writing data to a destination.
- InputStream and OutputStream are the basic stream classes in Java.
- Java encapsulates Stream under java.io package.
- Java defines two types of streams.
- The java.io package contains a large number of stream classes that provide capabilities for processing all types of data.

- These classes may be categorized into two groups based on the data type on which they operate.
  - Byte stream classes - These handle data in bytes (8 bits) i.e., the byte stream classes read/write data of 8 bits. Using these you can store characters, videos, audios, images etc.
  - Character stream classes - These handle data in 16 bit Unicode. Using these you can read and write text data only.

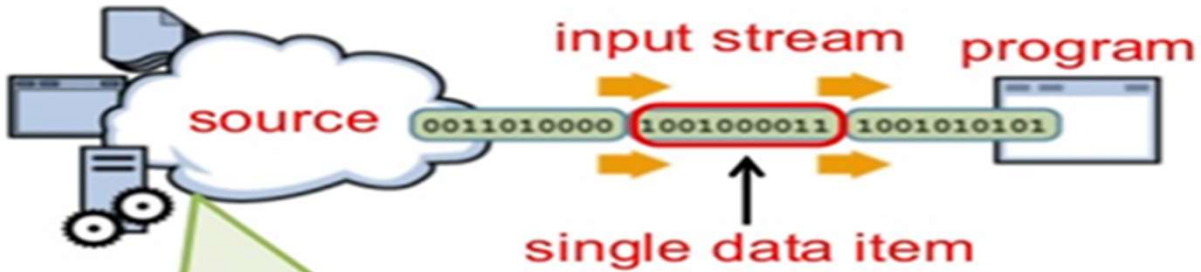

Java Stream Classes
Byte Stream Classes
(a) InputStream Classes
(b) OutputStream Classes
Character Stream Classes
(a) Reader Classes
(b) Writer Classes


Source
Inputstream
READ
JAVA
APPLICATION
Outputstream
Write
Destination

Input Stream Classes

- Java application uses an input stream to read data from a source; it may be a file, an array, peripheral device or socket.
- Input stream classes that are used to read bytes include a super class known as Inputstream and a number of subclasses for supporting various input-related functions.
- The super class InputStream is an abstract class, and, therefore, we cannot create instances of this class. Rather, we must use the subclasses that inherit from this class.

Input Streams



Source may be the keyboard, a file on disk, a physical device, another program, even an array or `String` in the same program.

- **OutputStream**
  - Java application uses an output stream to write data to a destination; it may be a file, an array, peripheral device or socket.
  - Output stream classes are derived from the base class Outputstream like InputStream, the OutputStream is an abstract class and therefore we cannot instantiate it.
  - The several subclasses of the OutputStream can be used for performing the output operations

Output Streams

Destination may be the console window, a file on disk, a physical device, another program, even an array or `String` in the same program.

The predefined stream

- All java program automatically import java.lang package and this package defines a class called System.
- Three Java Predefined streams or standard streams are available in the java.lang.System class.
- System.in
  - This is the standard stream for input.
  - This stream is used for reading data for the program from the keyboard by default

- **System.out**
  - This is the standard stream for output.
  - This stream is used for writing data from the program to an output device such as a monitor / console by default or to some specified file.
- **System.err**
  - This is a standard stream for error.
  - This is used to show an error message on the screen i.e. console by default for the users.

- System.in is an object of InputStream.
- On the other hand, System.out and System.err are both an object of type PrintStream.
- All these Java Predefined Streams are automatically initialized by Java's JVM (Java Virtual Machine) on startup.
- All these streams are byte streams but these are used to read and write character streams as well.

DataInputStream Class

- Java DataInputStream class allows an application to read primitive data from the input stream in a machine-independent way.
- It is called DataInputStream – because it reads data (numbers) instead of just bytes.
- This stream class wrap other stream classes for creating objects.
- Constructor is,
  - DataInputStream(InputStream obj)
  - Here obj is object of any other InputStream class.

- DataInputStream supports reading of primitive data type such as int, float, double,char, etc.
- Methods in DataInputStream class are,
  - int read(byte b[ ])- It is used to read the number of bytes from the input stream.
  - int read(byte b[ ], int off, int len)- It is used to read len bytes of data from the input stream
  - int readInt( )- It is used to read input bytes and return an int value.
  - byte readByte( )- It is used to read and return the one input byte.

- char readChar( )- It is used to read two input bytes and returns a char value.
- double readDouble( )- It is used to read eight input bytes and returns a double value.
- boolean readBoolean( )- It is used to read one input byte and return true if byte is non zero, false if byte is zero.
- String readLine( )- Reads the next line of text from the input stream. It reads successive bytes, converting each byte separately into a character, until it encounters a line terminator or end of file; the characters read are then returned as a String.

- Example program for add two numbers. The inputs are accepted from the keyboard

```
import  java.io.*;
class add
{
   public static void main( String args[ ] ) throws IOException
  {
              DataInputStream din= new DataInputStream(System.in);
             System.out.println("Enter first no:");
             int num1=Integer.parseInt(din.readLine ( ));
             int num2=Integer.parseInt(din.readLine ( ));
             int s=nm1+num2;
             System.out.println("Sum="+s);
  }
}
```

java.util package

- The package java.util contains a number of useful classes and interfaces.
- Although the name of the package might imply that these are utility classes, they are really more important than that.
- In fact, Java depends directly on several of the classes in this package.

- **Date class** :- java.util.Date class represents a specific instant in time, with millisecond precision.
- Example program,

```
import java.util.Date;
class DateDemo
    {
        public static void main(String args[ ])
        {
            // Instantiate a Date object
            Date date = new Date();
            // display time and date using toString()
            System.out.println(date);
        // Display number of milliseconds since midnight, January 1, 1970 GMT
            long msec = date.getTime();
            System.out.println("Milliseconds since Jan. 1, 1970 GMT = " + msec);
        }
    }
```

Scanner class

- Scanner is a class in java.util package used for obtaining the input of the primitive types like int, double, etc. and strings.
- It is the easiest way to read input in a Java program.
- The Java Scanner class breaks the input into tokens using a delimiter which is whitespace by default.
- It provides many methods to read and parse various primitive values.
- By the help of Scanner in Java, we can get input from the user in primitive types such as int, long, double, byte, float, short, etc.

- To get the instance of Java Scanner which reads input from the user, we need to pass the any input stream in the constructor of Scanner class.
- Constructors in Scanner class

1. **Scanner(InputStream source)**
   - This constructs a new Scanner that produces values scanned from the specified input stream.
   - For example,
     - Scanner in = **new** Scanner(System.in);

- Scanner class Methods to Take Input
  - Scanner class helps to take the standard input stream in Java.
  - we need some methods to extract data from the stream.
  - Methods used for extracting data are mentioned below:

Method	Description
nextBoolean()	Used for reading Boolean value
nextByte()	Used for reading Byte value
nextDouble()	Used for reading Double value
nextFloat()	Used for reading Float value
nextInt()	Used for reading Int value
nextLine()	Used for reading Line value
nextLong()	Used for reading Long value
nextShort()	Used for reading Short value
hasNext()	returns true if there is another token in the input. It's a blocking method and it will keep waiting for user input.

In the example below, we use different methods to read data of various types:

```
import java.util.Scanner;
class exampleScanner {
  public static void main(String[] args) {
    Scanner myObj = new Scanner(System.in);
    System.out.println("Enter name, age and salary:");
    // String input
    String name = myObj.nextLine();
    // Numerical input
    int age = myObj.nextInt();
    double salary = myObj.nextDouble();
    // Output input by user
    System.out.println("Name: " + name);
    System.out.println("Age: " + age);
    System.out.println("Salary: " + salary);
  }
}
```

Read some numbers from the console and print their sum

```
import java.util.Scanner;

public class ScannerDemo2 {
    public static void main(String[] args)
    {
        // Declare an object and initialize with predefined standard input object
        Scanner sc = new Scanner(System.in);
         // Initialize sum and count of input elements
        int sum = 0, count = 0;
         // Check if an int value is available
        while (sc.hasNextInt()) {
            // Read an int value
            int num = sc.nextInt();
            sum += num;
            count++;
        }
        System.out.println("No. of elements="+count);
         System.out.println("Sum ="+sum);
}
]
```